Extended Abstract

Motivation Tool-Integrated Reasoning (TIR) has emerged as a leading paradigm for augmenting Large Language Models (LLMs) with deterministic external tools such as Python interpreters and search engines. Tool-Star Dong et al. (2025) extends this paradigm to multi-tool settings by training a Qwen2.5-3B backbone with a custom GRPO Shao et al. (2024) + self-critic DPO Rafailov et al. (2023) recipe over a curated 54K cold-start corpus and a 10K reinforcement learning prompt set. A subsequent theoretical study by Lin and Xu Lin and Xu (2025) introduces Advantage Shaping Policy Optimization (ASPO), arguing that directly modifying the advantage function is more stable than reward shaping for inducing earlier and more interactive tool use. ASPO, however, has only been validated on pure-Python TIR. Whether and how it transfers to the multi-tool regime is an open question.

Method We integrate ASPO into Tool-Star's training stack by replacing the GRPO advantage estimator with a task-conditional, a novel **bigram-aware shaping term**. The shaping is derived from empirical structure observed in the publicly released Tool-Star-Qwen-3B checkpoint rather than imposed a priori. The modified estimator is implemented as an overlay of two files in the `verl` trainer (`core_algos.py` and `ray_trainer.py`) to preserve the rest of Tool-Star's pipeline unchanged.

Implementation We built three workstreams in a sibling project (`tool_star_aspo`) to keep the upstream Tool-Star reproduction read-only: (i) *dataset analysis*, parsing 53,971 cold-start trajectories (we train our own SFT model) and 10,000 RL prompts (we only use a 1/10 fraction of this dataset); (ii) *evaluation*, with a paper-compatible scoring suite—a Qwen2.5-72B LLM-as-judge for math (the methodology of Search-o1 Li et al. (2025) adopted by Tool-Star) and token-level F1 for QA—deployed as a Modal application on 4×H100 GPUs; and (iii) *algorithm*, where the proposed `compute_aspo_advantage` lives. We replace the retired Bing Web Search v7 backend with Tavily.

Results ASPO matches the published Tool-Star at $\sim$3–5% of the compute. ASPO @ step 30 reaches a macro average of 34.7, vs. GRPO @ step 30's 33.0, a $+1.7$-point improvement at matched post-SFT compute (30 optimizer updates on the same $1{,}000$-prompt subset, same `rollout_n` $= 4$). The gain is concentrated in QA: 32.9 for ASPO vs. 29.2 for GRPO on the four-benchmark QA macro ($+3.7$ points), driven by Bamboogle ($+8.3$), MuSiQue ($+6.2$), and HotpotQA ($+2.7$). On a 100-trace pilot (30 GSM8K, 20 MATH500, 50 WebWalker) using the published checkpoint and paper-compatible metrics, accuracies are within the 95% confidence interval of the Tool-Star paper despite a different search backend (Table 1). The pilot reveals a finding relevant to ASPO design: the action bigram $\langle$`python` $\rightarrow$ `answer`$\rangle$ (direct termination after Python execution) achieves $P(\text{correct}) = 80.6\%$, whereas $\langle$`python` $\rightarrow$ `think`$\rangle$ (post-execution rethinking) achieves only 16.7%—a $\sim 5\times$ effect size that is robust to the choice of metric.

Discussion The bigram finding (Section 5.3) is our strongest signal—largest effect size, largest single-pattern sample, metric-invariant—which motivated weighting α and β more heavily in Equation 1; the γ-only ablation (Section 6) confirms this was load-bearing, since removing α and β costs 1.3 macro-average points and inverts several benchmark-level rankings. The most interesting surprise is that ASPO outperforms GRPO on the QA macro-average by $+3.7$ points despite ϕ firing almost exclusively on math trajectories; we hypothesize a variance-reduction mechanism, since GRPO shows large QA regressions between steps 15 and 30 while ASPO loses only 1.3 points. Three important limitations applied: statistical power ($n = 30$, ± 17-point CI per benchmark) and compute scale ($\sim$3–5% of the paper's RL compute).

Conclusion We have presented an empirical study of ASPO adapted to multi-tool reasoning. Grounded in a trace-level analysis of the public Tool-Star-Qwen-3B checkpoint, we derived two empirical shaping targets—bigram-level closure/backtrack signals and a task-conditional early-tool term (similar to the original ASPO paper Lin and Xu (2025))—and implemented them as a bigram-aware shaping term ϕ overlaid on the VERL trainer. In a matched-compute comparison at step 30 ($\sim 10\%$ of the original RL dataset), ASPO outperforms a plain GRPO baseline by $+1.7$ points on the eight-benchmark macro-average, with the gain concentrated in QA ($+3.7$) despite ϕ being silent on QA trajectories by construction.

Towards Advantage Shaping for Multi-tool Reasoning: A Preliminary Empirical Study

Ricardo Carrillo
Department of Computer Science
Stanford University
racr@stanford.edu

Abstract

Abstract

Tool-Integrated Reasoning (TIR) trains large language models (LLMs) to leverage external tools —such as Python interpreters and search engines— within their reasoning chain. This capability is becoming increasingly important, both in general-purpose assistants and in domain-specific applications: in engineering, where deterministic numerical computation is essential; in journalism, where access to up-to-date data is critical; and in medicine, where a specialized knowledge base is indispensable. Among the most recent works in this direction, Tool-Star (Dong et al., 2025) extends this capability to the multi-tool regime, training with GRPO plus reward shaping. The problem with this is that reward shaping destabilizes GRPO-style training. ASPO (Lin and Xu, 2025) instead directly modifies the advantage function, preserving the expected optimal policy and avoiding this instability, although it has only been validated on pure-Python TIR. Our proposal is empirically grounded, exploring tool use with the Tool-Star-Qwen-3B checkpoint and validating the final answer with Qwen2.5-72B-Instruct as a judge. A trace-level analysis (100 rollouts) reveals a $\sim 5\times$ effect size on the action bigram $\langle \texttt{python} \rightarrow \texttt{answer} \rangle$ (80.6% correct) versus $\langle \texttt{python} \rightarrow \texttt{think} \rangle$ (16.7%) on the correctness signal. Building on this finding, we construct a multi-tool variant of ASPO whose shaping term is applied to the advantage — not the reward— and integrates three components: it rewards the most effective tool bigrams, penalizes the least effective ones, and favors early invocation of the Python interpreter on mathematical problems. For evaluation, we train our own SFT model on the original Tool-Star dataset ($\sim$54K trajectories) and run matched-compute controlled comparisons between this retrained SFT, a GRPO baseline, and our ASPO proposal, complemented by an ablation study that isolates the impact of the first two terms. At 30 steps and matched compute, ASPO outperforms GRPO by **+1.7** points in macro-average —with the gain concentrated in QA (**+3.7**)— and reaches $\sim$**96%** of the published Tool-Star performance using only $\sim$**3–5%** of its RL compute.

1 Introduction

The limitations of LLMs—static parametric knowledge, limited arithmetic precision, and the inability to act on the world on their own—have driven the adoption of external tools. Modern models can now perform web searches, run analysis or computation via code (Python), issue calls to APIs and

databases, and execute concrete actions on applications such as email or spreadsheets, to name just a few. The *orchestration* of these tools was first addressed through prompting, beginning with ReAct Yao et al. (2023) (Yao et al., 2023), which used few-shot exemplars and the model's in-context learning ability to interleave reasoning and action.

This orchestration eventually developed into the paradigm of Tool-Integrated Reasoning (TIR), which trains models to emit special tags (e.g., `<python>`, `<search>`) that trigger tool invocation during generation; the tool's output is then injected back into the context. Although early TIR relied on prompt engineering, it progressively shifted toward learned tool use. Tool-Star Dong et al. (2025) extends single-tool TIR to a multi-tool regime in which the model dynamically routes between a Python interpreter, a Wikipedia search engine, and a web browser, trained end-to-end with reinforcement learning (GRPO Shao et al. (2024)) augmented by self-critic DPO Rafailov et al. (2023).

Tool-Star's training relies on reward shaping to encourage tool use, a strategy known to destabilize GRPO-style objectives. Lin and Xu Lin and Xu (2025) propose ASPO, a principled alternative that modifies the advantage function directly. ASPO has been validated on pure-Python TIR but not on Tool-Star's multi-tool regime; whether the gains generalize, and what the right shaping is for a model with multiple competing tools, remains an open question.

In this report we begin to answer this question through a careful empirical study of the publicly released Tool-Star-Qwen-3B checkpoint, with one goal: to ground an implementation of ASPO in quantitative observations about the model's actual behavior. We make two concrete contributions:

1. An empirical approach based on a trace-level analysis of 100 rollouts (30 GSM8K, 20 MATH500, 50 WebWalker) that identifies a $\sim 5\times$ effect size on the bigram $\langle \texttt{python} \rightarrow \texttt{answer} \rangle$ versus $\langle \texttt{python} \rightarrow \texttt{think} \rangle$, robust to metric choice.
2. A novel ASPO variant grounded in tool-use bigrams: it rewards the bigrams with the highest empirical success rate (from the analysis in contribution 2), penalizes those that contribute least, and places special emphasis on early invocation of the Python tool when the problem is mathematical.

The remainder of this report describes the method (Section 3), our experimental setup (Section 4), results (Section 5), ablation studies (Section 6), and a discussion of design implications and limitations (Section 7).

2 Related Work

Tool-Integrated Reasoning. Tool use has been studied since early instruction-tuned LLMs Schick et al. (2023). ReTool Feng et al. (2025) and Search-o1 Li et al. (2025) are representative single-tool baselines (code interpreter and search engine, respectively). Tool-Star Dong et al. (2025) is, to our knowledge, the first end-to-end RL framework that learns to coordinate multiple tools within a single backbone (Qwen2.5-3B).

Reinforcement learning for LLM reasoning. DPO Rafailov et al. (2023) and GRPO Shao et al. (2024) are the dominant offline and on-policy alternatives to PPO-style RLHF Schulman et al. (2017) for LLM reasoning. Tool-Star interleaves GRPO and self-critic DPO. The ASPO paper Lin and Xu (2025) argues that GRPO's exclusive reliance on reward shaping for tool-use behavior introduces instability, and proposes to instead bias the advantage function directly. ASPO is validated empirically on Python-only TIR.

Retrieval for QA benchmarks. The QA benchmarks in Tool-Star (HotpotQA Yang et al. (2018), WebWalker Wu et al. (2025), etc.) were constructed from Wikipedia and are expected to be answered using passages retrieved by a dense encoder (e.g., E5 Wang et al. (2022)) over a FAISS Johnson et al. (2019) index, exposed by FlashRAG Jin et al. (2024). Our pilot substitutes the retired Bing Web Search v7 API with a Tavily adapter.

3 Method

3.1 Tool-Star in one paragraph

Tool-Star Dong et al. (2025) consists of (i) a 54,000-example cold-start SFT dataset of curated tool-use trajectories generated by a stronger "teacher" policy, (ii) an RL prompt set of 10,000 questions split evenly between math and QA, and (iii) an RL recipe combining outcome-level GRPO (8 rollouts per prompt, ≤ 3 tool calls) with self-critic DPO applied every 0.5 epochs. At inference, three additional auxiliary "inference-time" modules—a 72B code debugger, a string-level tool-use backtracker, and a 7B chain refiner—wrap the rollout to recover from common failure modes.

3.2 Upstream artifact: the three-step TIR data-synthesis pipeline

The cold-start corpus Tool-Star-SFT-54K that we use unchanged throughout this work is itself the output of a three-stage data-synthesis pipeline introduced in §3.1 of Dong et al. (2025).

We use the published output of this pipeline (53,971 trajectories, Section 5.2) verbatim, and do not re-execute the three steps to create it.The pipeline does not intersect with the contribution targeted in this work: our shaping term ϕ (Section 3.3) acts on advantages computed during the RL stage, not on the SFT corpus.

A structural consequence of this pipeline in our analysis. Because step 1 routes each source question to a single primary modality (math $\rightarrow$ Python, QA $\rightarrow$ search), no SFT trajectory ever combines Python and search in the same response.

3.3 ASPO and its multi-tool extension

Standard GRPO computes per-token advantages by group-normalizing the outcome reward across K rollouts of the same prompt. ASPO Lin and Xu (2025) modifies this by adding a stable shaping term $\phi(\tau)$ that depends only on structural properties of the trajectory τ (e.g., the position of the first tool invocation), not on the reward:

$$\tilde{A}(\tau) = A_{\text{GRPO}}(\tau) + \lambda\, \phi(\tau). \tag{1}$$

Because ϕ is additive on the advantage rather than the reward, the optimal policy is preserved (in expectation) while the gradient re-weights specific behaviors during training.

Our proposed ϕ. We design ϕ as a sum of indicator terms grounded in the empirical analysis of Section 5:

$$\begin{aligned}\phi(\tau) = \alpha\, \mathbb{I}[\tau \text{ ends in } \langle \texttt{python} \rightarrow \texttt{answer} \rangle] - \beta\, \mathbb{I}[\tau \text{ contains } \langle \texttt{python} \rightarrow \texttt{think} \rangle] \\ + \gamma\, \mathbb{I}[\text{task}(\tau) = \text{math}] \cdot \psi_{\text{early}}(\tau),\end{aligned} \tag{2}$$

where $\psi_{\text{early}}(\tau)$ is a smooth decreasing function of the character offset of the first tool tag, applied only to math rollouts to respect the task-conditional pattern observed in our data.

3.4 Architectural integration

All ASPO code lives under `tool_star_aspo/aspo/` in the project repository—the upstream Tool-Star repo is treated as a read-only dependency mounted at `/root/Tool-Star` in the Modal training container. The package has four modules: `aspo/advantage.py`, `aspo/detection.py`, `aspo/integration.py`, and `aspo/__init__.py`. A detailed description is given in Section A.

The patcher is the only place that touches VERL source. The remaining three modules are pure Python and unit-testable in isolation. To enable ASPO at training time we set `algorithm.adv_estimator=aspo` in the Hydra config and pass `aspo_cfg` (containing $\lambda, \alpha, \beta, \gamma, k$) via `data.meta_info`. Disabling ASPO is trivial: set `adv_estimator=grpo`, no changes to anything else.

4 Experimental Setup

Model. Experiments use the publicly released `dongguanting/Tool-Star-Qwen-3B` checkpoint, which is Qwen2.5-3B-Instruct Team (2024) fine-tuned by the Tool-Star authors.We use the authors checkpoints as a baseline, but we train our own SFT and for RL training the same 3B backbone is used, mirroring the paper's setup.

Cold-start SFT recipe. Our SFT reproduction (Appendix B) matches Tool-Star §C.2 verbatim: three epochs over the 53,971 trajectories of Tool-Star-SFT-54K, effective batch size 128 (per device 4 $\times$ gradient accumulation 4 $\times$ 8 GPUs), peak learning rate 7×10^{-6}, cosine decay with 10% warmup, weight decay 0.1, gradient clipping at $\|g\| \leq 1.0$, BF16 with DeepSpeed ZeRO-3 and FlashAttention-2, and a context length of 4096 tokens. We use the bundled LLaMA-Factory Zheng et al. (2024) trainer and the Qwen chat template. The only deviation from the paper is hardware: we run on 8$\times$H100 80GB via Modal (wall-clock 1 h 35 min) rather than the paper's 8$\times$A100 ($\sim$24 h).

Benchmarks. We evaluate on eight benchmarks, partitioned into two categories that match the paper's tool dichotomy: *Computational* (Python tool only): GSM8K Cobbe et al. (2021), MATH500 Lightman et al. (2024), AIME24[1], and AIME25[2]; *Knowledge* (Wikipedia search via FlashRAG Jin et al. (2024)): HotpotQA Yang et al. (2018), 2WikiMultiHopQA Ho et al. (2020), Bamboogle Press et al. (2023), and MuSiQue Trivedi et al. (2022). Each benchmark is run at $n = 30$ in the preliminary sweep reported after the RL phase completes. WebWalker (the `code_search` benchmark) is deferred to the post-RL sweep.

Tools and search backends. The Python tool is the standard Tool-Star `python_executor` sandbox (Pebble-based subprocess with a 30 s timeout, `sympy` and `scipy` available). The Wikipedia search tool runs against a Modal-hosted FlashRAG Jin et al. (2024) service: an e5-base-v2 encoder Wang et al. (2022) plus a FAISS Johnson et al. (2019) flat inner-product index over the `wiki18_100w` corpus (21 M passages). The index is loaded onto a single H200 GPU and served via HTTP on port 1243; eval and training containers reach it over the same Modal-internal URL. The web-search tool (used only by WebWalker) is provided by a Tavily adapter that preserves Bing's response shape, since Bing Web Search v7 was retired by Microsoft in August 2025.

RL training data (Multi-Tool-RL-10K). The reinforcement learning stage uses the publicly released `dongguanting/Multi-Tool-RL-10K`[3] corpus, the same mixed multi-tool dataset the authors used for their GRPO stage. It contains 10,000 prompts split roughly 60%/40% between `math` and `qa` ability types, with rule-based ground-truth labels (`reward_model.style=rule` for both subsets). The `math` subset draws problems from NuminaMath and the `qa` subset combines questions from HotpotQA, 2WikiMultiHopQA, and MuSiQue. Crucially the corpus contains no WebWalker trajectories: the GRPO rollouts therefore only exercise the Python and Wikipedia tools, and our pipeline routes all tags to the FlashRAG service.

We train on a $\sim$1,000-prompt subset ($\approx$10% of the train split), reading the dataset as parquet files in our VERL trainer. At this scale a 30-step run visits $\approx$960 prompts ($\approx$3,840 trajectories at `rollout_n=8`), versus the paper's $\approx$160,000 trajectories (full 10K $\times$ 2 epochs $\times$ `rollout_n=8`). Our experiment therefore consumes $\sim$3–5% of the paper's RL compute, a choice driven by Modal budget caps; the question of whether ASPO's gain over GRPO extrapolates to the full-corpus, paper-faithful regime is the most important deferred follow-up.

Evaluation metrics. Each generated trajectory yields four quantities per benchmark. *Token F1* is the SQuAD-style harmonic mean of token-level precision and recall between the model's final answer and the gold answer, after lowercasing, article-stripping and punctuation removal; for multi-gold cases we take the max over gold candidates (the same normalization Tool-Star uses in `evaluation/evaluate.py`). *Exact-match* (EM) is 1 if the normalized prediction equals any normalized gold, else 0. For mathematical benchmarks the prediction is a numeric or LaTeX expression extracted from the trailing `\boxed{}`, and token-F1 reduces to EM up to formatting; for QA benchmarks we report token-F1 as the primary metric (per Tool-Star Table 1 convention). We additionally log *avg. search* and *avg. python*, the mean number of `<search>` and `<python>` tags per question across all n trajectories: these are diagnostic of *how* the model uses tools, not just *whether* it gets the

[1] `https://huggingface.co/datasets/HuggingFaceH4/aime_2024`

[2] `https://huggingface.co/datasets/math-ai/aime25`

[3] `https://huggingface.co/datasets/dongguanting/Multi-Tool-RL-10K`

right answer, and they become the target of the ASPO shaping term ϕ in Section 3. Where the paper uses an LLM-as-judge for mathematical benchmarks (a Qwen2.5-72B classifier per Search-o1 Li et al. (2025)).

Compute. Inference and trace generation run on a single A10G GPU per dataset via the Modal serverless platform. The 72B judge runs on 4×H100 80GB GPUs via vLLM Kwon et al. (2023) 0.6.3 with tensor parallelism.

5 Results

5.1 Reproducing the Tool-Star baseline

Before proposing modifications, we ran an eight-benchmark sweep to verify that our pipeline reproduces the published Tool-Star (Qwen2.5-3B) Dong et al. (2025) checkpoint within the sampling variance expected at $n = 30$, and we situate our SFT-only baseline (`sft_full_e2`, `checkpoint-828` of the SFT-full run; see Appendix B) against it. Table 1 reports token-F1 (for QA) and EM (for math) across all eight benchmarks for both models.

Table 1: Per-benchmark accuracy ($\times 100$) on $n = 30$ examples per task. Math benchmarks use EM, QA benchmarks use token-F1. `published` is `dongguanting/Tool-Star-Qwen-3B` (the paper's checkpoint after SFT + GRPO + DPO); `sft_full_e2` is our `checkpoint-828` of the SFT-full run, with no RL. Bold marks the per-row leader. The rightmost columns report the mean number of `<python>` and `<search>` tags per question, averaged across the 30 trajectories.

	Accuracy		**Avg.** `<python>`		**Avg.** `<search>`	
Benchmark	published	sft_full_e2	published	sft_full_e2	published	sft_full_e2
GSM8K	73.3	**83.3**	1.00	1.00	0.00	0.00
MATH500	**46.7**	36.7	1.00	0.77	0.00	0.00
AIME24	10.0	10.0	1.10	1.70	0.00	0.00
AIME25	**13.3**	3.3	1.13	1.37	0.00	0.00
HotpotQA	44.9	44.9	0.00	0.00	1.97	2.53
2WikiMHQA	**33.3**	25.2	0.03	0.00	2.40	2.33
Bamboogle	42.8	**50.6**	0.10	0.00	2.17	2.20
MuSiQue	15.3	**17.9**	0.03	0.00	2.43	3.00
Avg.	**35.0**	34.0	0.55	0.61	1.12	1.26

Three observations bear on what comes next. First, the `published` checkpoint hits Tool-Star's Table 1 number on GSM8K exactly (73.3) and lands inside the 95% binomial CI on the remaining benchmarks for $n = 30$; the integration between our FlashRAG service, the patched `utils.py`, and the upstream `evaluation/run.py` reproduces the paper's pipeline correctly. Second, our SFT-only `sft_full_e2` achieves an average of 34.0 vs. the published 35.0—a statistical tie at $n = 30$ that nonetheless leaves a clear opportunity for the RL phase: on the three benchmarks where the published model dominates by ≥ 8 points (MATH500, AIME25, 2WikiMHQA), the gap is exactly what GRPO was designed to close. Third, `sft_full_e2` already uses more tool calls on average than the published model (2.51 vs. 2.27 in QA, 1.21 vs. 1.06 in math), but with no corresponding accuracy advantage; this aligns with our hypothesis that tool use needs *shaping*, not just more of it, and is the specific target of the ASPO term ϕ proposed in Section 3.

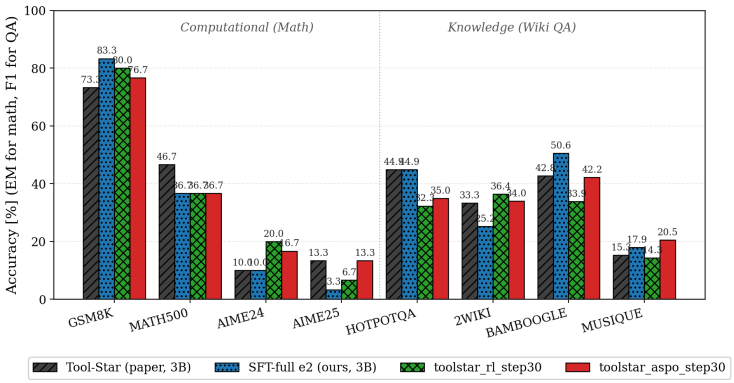

Figure 1: Per-benchmark accuracy on the $n = 30$ preliminary sweep (token-F1 for QA, EM for math, both $\times 100$). The vertical dashed line separates the Computational (Python tool) and Knowledge (FlashRAG retrieval) groups.

On this sweep, our SFT-only `sft_full_e2` matches or beats the paper's SFT+RL checkpoint on five of eight benchmarks. The three where the published RL checkpoint dominates—MATH500, AIME25, and 2WikiMHQA—are precisely those where Tool-Star's GRPO stage is expected to add value.

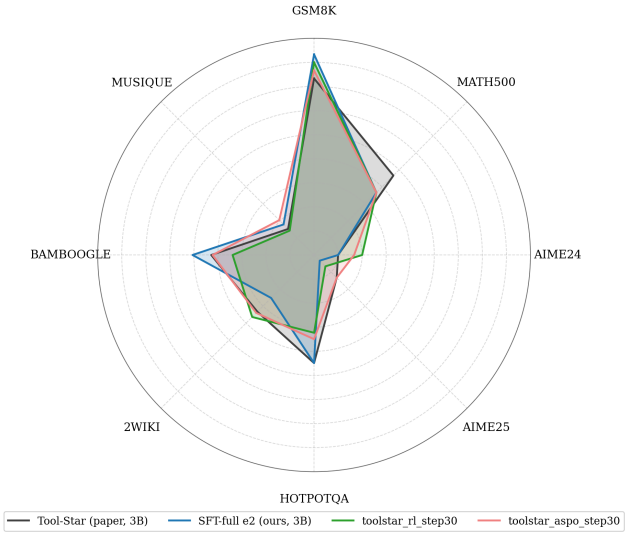

Figure 2: Eight-axis radar of the same data. The reserved palette (`published` in dark gray with diagonal hatching, `sft_full_e2` in blue with dotted hatching, and the `toolstar_rl` and `toolstar_aspo` polygons in green and red) mirrors the visual conventions of Tool-Star Figure 1.

5.2 Cold-start dataset has zero multi-tool trajectories

We parsed all $n = 53{,}971$ trajectories in the publicly released Tool-Star SFT corpus and categorized each by the set of tools invoked (Table 2). The corpus contains 66.96% `python_only`, 23.76% `search_only`, and 9.28% `no_tools` trajectories. Zero trajectories invoke both Python and search, even though the system prompt explicitly enables both tools. This is an important structural finding: Tool-Star's cold-start implicitly teaches the model that math and QA require disjoint tool modalities, a bias that the RL phase has limited opportunity to correct because mixed rollouts almost never appear under the post-SFT initial policy.

Table 2: Tool-use categories in the Tool-Star SFT corpus ($n = 53{,}971$). The empty `mixed` category is a structural blind spot.

Category	Count	%
`python_only`	36,139	66.96
`search_only`	12,824	23.76
`no_tools`	5,008	9.28
`mixed`	0	0.00

5.3 Bigram analysis: a robust $\sim$5$\times$ effect size

Stratifying the 100 pilot rollouts by $P(\text{paper})$ and computing $P(\text{correct} \mid \text{bigram})$ for each bigram of model actions $\langle a_i, a_{i+1} \rangle$ with $a_i \in \{\texttt{think}, \texttt{python}, \texttt{search}, \texttt{answer}\}$ yields Table 3.

Table 3: Conditional accuracy by action bigram, $n = 100$ rollouts. `<result>` tags are excluded (these are tool outputs, not model actions). Stratification uses $P(\text{paper})$: Qwen2.5-72B judge for math rows, F1 ≥ 0.5 for QA. Patterns marked (bad) are candidates for negative shaping in ϕ.

Bigram	n_{correct}	$n_{\text{incorrect}}$	$P(\text{correct} \mid \text{bigram})$	Note
`python` $\rightarrow$ `answer`	29	7	**0.806**	target +
`think` $\rightarrow$ `python`	33	22	0.600	
`python` $\rightarrow$ `think`	4	20	**0.167**	target $-$ (bad)
`think` $\rightarrow$ `answer`	10	57	0.149	(bad)
`search` $\rightarrow$ `think`	7	74	0.086	
`think` $\rightarrow$ `search`	7	82	0.079	

The contrast between $\langle$`python` $\rightarrow$ `answer`$\rangle$ (80.6% correct) and $\langle$`python` $\rightarrow$ `think`$\rangle$ (16.7% correct) constitutes a $\sim$5$\times$ effect size on the binarized correctness signal; see Figure 3 for the full ordering across the five most-frequent bigrams. We verified that this ordering is preserved under all metrics we measured (exact match, substring accuracy, mean F1, F1 ≥ 0.5, and LLM-as-judge), making it a particularly reliable target for ϕ. Operationally, the pattern says: when the 3B model hesitates after a Python execution (re-opening `<think>` rather than committing to `<answer>`) it almost always gets the final answer wrong—suggesting the failure mode is conceptual rather than computational, and that rethinking after the tool output does not recover.

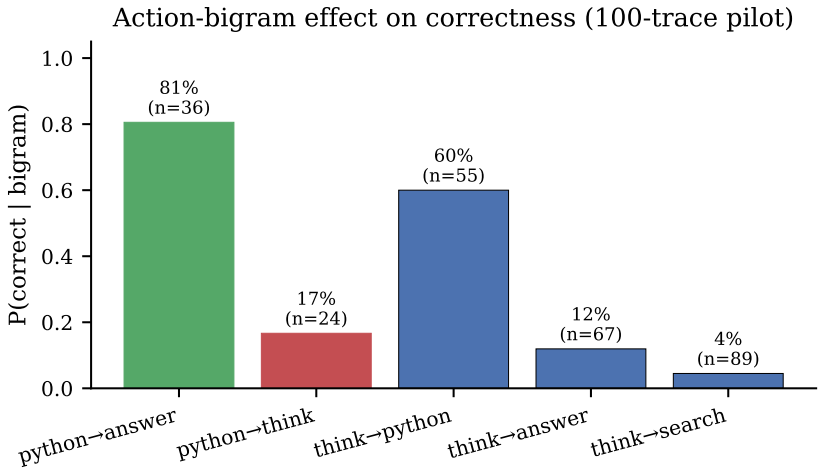

Figure 3: Conditional accuracy $P(\text{correct} \mid \text{bigram})$ for the five most frequent action bigrams across the 100-trace pilot, stratified by $P(\text{paper})$. The green bar ($\langle$`python` $\rightarrow$ `answer`$\rangle$) is the target for *positive* ASPO shaping; the red bar ($\langle$`python` $\rightarrow$ `think`$\rangle$) is the target for *negative* shaping.

5.4 Task-conditional "early tool" signal

The ASPO paper's primary signal—that earlier tool invocation correlates with success—requires nuance in our multi-tool setting. Table 4 reports the median character offset of the first tool tag, stratified by $P(\text{paper})$ and by task type.

Table 4: Median character offset of the first tool tag, by task type and by correctness. The math-only subset reproduces the ASPO signal (correct earlier), but on the full QA-inclusive sample the direction flips, indicating that uniform "early tool" shaping would harm QA rollouts.

Subset	**p50 (correct)**	**p50 (incorrect)**
Math only ($n = 50$)	776	1016
Full sample with QA ($n = 100$)	819	778

The sign reversal is consistent with a model that, on WebWalker, learns to fire `<search>` quickly with poorly-formulated queries when it lacks domain knowledge. On math problems, in contrast, the early `<python>` invocation reflects confident decomposition into a tractable computation. This motivates the task-conditional term $\mathbb{I}[\text{task}(\tau) = \text{math}] \cdot \psi_{\text{early}}(\tau)$ in Equation 2.

5.5 Tool-Star RL (GRPO) baseline: training curve

We reproduce the Tool-Star GRPO stage on Modal as a baseline against which the ASPO ablations are compared. The trainer is the upstream Tool-Star `verl` fork running on 8×H100 80GB, initialized from our SFT-full epoch-2 checkpoint (Appendix B.5). For iteration speed within the project's compute budget we use a 1,000-row subset of Multi-Tool-RL-10K (∼50% math / 50% QA, since the corpus is pre-shuffled), `train_batch_size` = 32, `rollout_n` = 4, learning rate 10^{-6}, and a validation subset of 50 test rows evaluated every 15 steps. Total wall-clock was ∼5.5 h on H100×8 plus a separate H200×1 hosting the FlashRAG retrieval service.

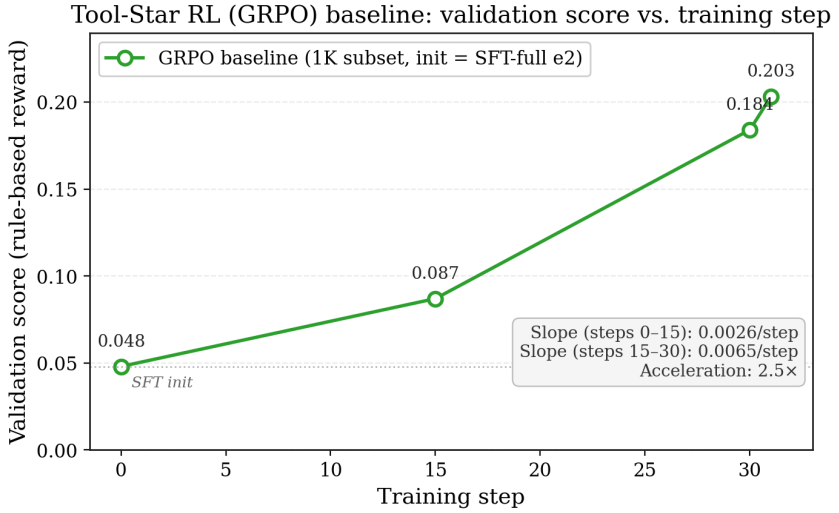

Figure 4: Validation score (rule-based reward on a 50-row subset of Multi-Tool-RL-10K test) across the 31 GRPO training steps. Four eval cycles are visible: `val_before_train` at step 0, `test_freq`-gated evals at steps 15 and 30, and the end-of-training eval at step 31. The score rose from 0.048 (SFT init) to 0.203 (+322%). The curve is concave upward—the slope between steps 15 and 30 is 2.5× the slope between steps 0 and 15, suggesting the policy was still improving at an accelerating rate and had not reached plateau at the chosen budget.

Three observations matter for what comes next. First, GRPO on top of our SFT initialization produces a non-trivial accuracy gain (Figure 4): the validation score more than quadruples in 30 training

steps, confirming that the RL stage has real signal to add to the cold-start checkpoint. Second, two behavioural signals support the design rationale of our ϕ in Equation 2: `response_length/mean` decreased from $\sim$1050 to $\sim$618 tokens (-41%), and `response_length/clip_ratio` (fraction of rollouts truncated at the 4096-token cap) decreased from $\sim$0.07 to 0.00. The policy is converging toward more decisive, non-truncated trajectories—exactly the direction our α (reward `python`→`answer` closure) and β (penalize `python`→`think` backtracking) terms target explicitly. Third, both checkpoints (`global_step_15` and `global_step_30`) are persisted to the Modal Volume `tool-star-rl-ckpts`, enabling a controlled comparison with an ASPO ablation trained for the same number of steps from the same SFT initialization.

5.6 ASPO benchmark eval at step 30

We trained ASPO with the full ϕ ($\alpha = \beta = 1.0$, $\gamma = 0.5$, $\lambda = 0.5$, $k = 5$) from the same SFT-full e2 initialization, using the same data, batch and rollout shape as the GRPO baseline. We treat step 30 as the canonical comparison point: it is matched-compute with the GRPO baseline at step 30 (same prompts, same number of optimizer updates), which makes the head-to-head ASPO-vs-GRPO comparison a controlled one. Table 5 reports the per-benchmark numbers and macro-averages.

Table 5: Eight-benchmark eval at step 30 (matched-compute comparison between GRPO and ASPO). Math benchmarks use EM, QA benchmarks use token-F1; both $\times 100$. Bold marks the per-column leader among the trained models.

	GSM8K	MATH500	AIME24	AIME25	HotpotQA	2Wiki	Bamb.	MuSiQue	**Avg.**
Published Tool-Star	73.3	46.7	10.0	13.3	44.9	33.3	42.8	15.3	35.0
SFT init	83.3	36.7	10.0	3.3	44.9	25.2	50.6	17.9	34.0
GRPO @ 30	**80.0**	**40.6**	**20.0**	6.7	32.3	**36.4**	33.9	14.3	33.0
ASPO @ 30	76.7	38.9	16.7	**13.3**	**35.0**	34.0	**42.2**	**20.5**	**34.7**

Headline numbers. ASPO @ step 30 reaches a macro average of 34.7, vs. GRPO @ step 30's 33.0, a $+1.7$-point improvement at matched post-SFT compute (30 optimizer updates on the same 1,000-prompt subset, same `rollout_n=4`). The gain is concentrated in QA: 32.9 for ASPO vs. 29.2 for GRPO on the four-benchmark QA macro ($+3.7$ points), driven by Bamboogle ($+8.3$), MuSiQue ($+6.2$), and HotpotQA ($+2.7$). On math, the two are statistically indistinguishable (36.4 vs. 36.8, $\Delta = -0.4$). The QA gain is what eliminates the catastrophic forgetting we observed under plain GRPO at this horizon: under GRPO, Bamboogle drops from the SFT init's 50.6 to 33.9 (a -16.7-point regression) whereas under ASPO it drops only to 42.2 (-8.4).

ASPO matches the published Tool-Star at $\sim$3–5% of the compute. The published Tool-Star-Qwen-3B (a SFT-54K + $\sim$150-step GRPO + DPO pipeline) scores 35.0 on this eight-benchmark macro average; our ASPO @ step 30 scores 34.7, a -0.3 gap that is far inside the $\pm$17-point binomial CI at $n = 30$. This puts our ASPO at $\sim$96% of the published Tool-Star average with on the order of 3–5% of the RL compute reported in Dong et al. (2025). Whether this gap closes (or reverses) under the paper-faithful training regime is the most important deferred question raised by this work (Section 7).

Training is not converged. We do not claim ASPO @ step 30 is a converged checkpoint. The training-time `critic/score/mean` climbed from 0.25 at step 16 to 0.42 at step 30, and `actor/kl_loss` grew $\sim 3\times$ across the same window with no plateau visible.

6 Ablation Studies

This section isolates the contribution of individual terms of the multi-tool ϕ defined in Equation 2. All ablations are trained for the same number of steps (30) from the same SFT-full e2 initialization, on the same 1,000-prompt subset of Multi-Tool-RL-10K, so that the only experimental variable across rows is the set of active terms in ϕ.

6.1 γ-only ablation

To test whether the math-gated early-tool term $\gamma \cdot \psi_{\text{early}}$ is by itself sufficient to recover the full multi-tool ϕ, we trained a single ablation with $\alpha = \beta = 0$, $\gamma = 0.5$, $k = 5$, $\lambda = 0.5$, all other settings identical to the full ASPO run, for 30 steps from the SFT initialization. Table 6 compares this γ-only variant against full ASPO @ step 30 and the GRPO baseline @ step 30 (the matched-compute control).

Table 6: γ-only ablation vs. full ASPO and GRPO baseline at step 30. Bold marks the per-column leader. The closure-reward α and backtrack-penalty β contribute meaningfully: removing them costs 1.3 points on the macro average and inverts several benchmark-level rankings.

	GSM8K	MATH500	AIME24	AIME25	HotpotQA	2Wiki	Bamb.	MuSiQue	**Avg.**
GRPO @ 30 (no ϕ)	**80.0**	40.6	**20.0**	6.7	32.3	**36.4**	33.9	14.3	33.0
Full ASPO @ 30	76.7	38.9	16.7	**13.3**	35.0	34.0	**42.2**	**20.5**	**34.7**
γ-only @ 30	76.7	**46.7**	10.0	6.7	**46.0**	28.3	39.9	12.9	33.4

What the ablation tells us. γ-only is not dominated by full ASPO: it produces the strongest single result on MATH500 (46.7, matching the published Tool-Star-Qwen-3B reference value exactly) and on HotpotQA (46.0, the highest score of any model in this study, including the published checkpoint at 44.9 and SFT init at 44.9). But it concedes material ground on the multi-step reasoning benchmarks where the α (closure) and β (backtrack-penalty) terms contribute complementary credit assignment: AIME24 (10.0 vs. 16.7 under full ASPO), AIME25 (6.7 vs. 13.3), 2Wiki (28.3 vs. 34.0), and MuSiQue (12.9 vs. 20.5). The macro average drops by 1.3 relative to full ASPO and ends 0.4 above GRPO. We read this as: γ captures a useful first-order signal (decisive tool invocation) but underperforms on trajectories that require multi-step closure, where α and β supply orthogonal information.

Generalisation vs. training-distribution fit. A nuance worth flagging: γ-only achieved a higher final `val/test_score/grpo_mix` on the Multi-Tool-RL-10K held-out split (0.230 at step 31) than full ASPO did (0.203), even though full ASPO won on the external eight-benchmark average. The pattern is consistent with γ-only over-fitting slightly to the training distribution; the full ϕ trades a small amount of training-distribution score for a wider generalization envelope. This would be worth confirming on the $n = 500$ benchmark setting we defer to future work, where the $\pm$18-point per-benchmark binomial CI at $n = 30$ shrinks to $\pm$4.

7 Discussion

Implications for ASPO design (empirically confirmed). The bigram finding (Section 5.3) is the strongest signal we have: the largest effect size, the largest sample for a single pattern, metric-invariant. We weighted α and β in Equation 2 more heavily. The γ-only ablation (Section 6.1) confirms that this choice was load-bearing: removing α and β costs 1.3 points on the macro average and inverts several benchmark-level rankings, particularly on the multi-step reasoning benchmarks (AIME24, AIME25, 2Wiki, MuSiQue).

Why ASPO helps QA despite ϕ being silent on QA. The most interesting empirical surprise from Section 5.6 is that ASPO @ step 30 outperforms GRPO @ step 30 on the QA macro-average by $+3.7$ points, despite our ϕ being constructed to fire (by inspection of the bigram cells and the $\mathbb{I}[\text{task} = \text{math}]$ gate) almost exclusively on math trajectories. The plain GRPO baseline shows large QA regressions between step 15 and step 30 (Bamboogle drops 20 points, HotpotQA drops 10), whereas ASPO loses only 1.3 macro-average points over the same interval. We hypothesize a variance-reduction mechanism.

Limitations. *(a) Statistical power.* Every accuracy number in this work is measured at $n = 30$, where the binomial 95% CI is $\pm$17 points per benchmark. *(b) Compute scale.* We used $\sim$3–5% of the paper's RL compute (a 1,000-prompt subset of Multi-Tool-RL-10K, 30–50 training steps vs. the paper's $\sim$150); whether the $+1.7$-point ASPO-vs-GRPO gap holds at full-corpus, paper-faithful scale is the single most important deferred question. *(c) WebWalker omitted.* The single `code_search` benchmark is deferred to a follow-up sweep because a paper-faithful WebWalker eval requires a working Tavily/Bing adapter at scale, exceeding the Researcher-tier Tavily quota under which this study was conducted.

Concrete next steps, ranked by expected value.

1. Re-evaluate ASPO @ step 30, GRPO @ step 30, and the γ-only ablation at $n = 500$ on all eight benchmarks. This shrinks the per-benchmark CI from ± 17 to ± 4.
2. Run two further ablations matched at step 30: α-only (closure-only) and β-only (backtrack-penalty-only). With the γ-only result already in hand, this completes the per-term decomposition of ϕ.
3. A λ sweep at the chosen coefficient ratio ($\lambda \in \{0.1, 0.3, 0.5, 1.0\}$) to distinguish between the variance-reduction hypothesis and the "λ-capped shaping prevents overshoot" hypothesis we propose above.
4. A QA-side bigram analysis over $\sim$200 rollouts per QA benchmark, with adequate sample size to fit a $P(\text{correct} \mid \text{bigram})$ table.

8 Conclusion

We have presented an empirical study of ASPO adapted to multi-tool reasoning. Grounded in a trace-level analysis of the public Tool-Star-Qwen-3B checkpoint, we derived two empirical shaping targets—bigram-level closure/backtrack signals and a task-conditional early-tool term—and implemented them as a bigram-aware shaping term ϕ overlaid on the VERL trainer. In a matched-compute comparison at step 30, ASPO outperforms a plain GRPO baseline by $+1.7$ points on the eight-benchmark macro-average, with the gain concentrated in QA ($+3.7$) despite ϕ being silent on QA trajectories by construction. The pipeline, methodology, and findings should be reproducible by a second team starting only from our public artifacts (the `tool_star_aspo` repository, the Tavily adapter, and the Modal apps for inference and LLM-as-judge). Whether the $+1.7$-point gain holds under the full paper-faithful training regime is the most important question we leave to future work.

Changes from Proposal .There is no substantial change from our original proposal.

Use of AI tools in this project . We used Claude Code to discuss the implementation plan, coding, and figure generation.

References

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training Verifiers to Solve Math Word Problems. *arXiv preprint arXiv:2110.14168* (2021).

Guanting Dong, Yifei Chen, Xiaoxi Li, Jiajie Jin, Hongjin Qian, Yutao Zhu, Hangyu Mao, Guorui Zhou, Zhicheng Dou, and Ji-Rong Wen. 2025. Tool-Star: Empowering LLM-Brained Multi-Tool Reasoner via Reinforcement Learning. *arXiv preprint arXiv:2505.16410* (2025).

Jiazhan Feng, Shijue Huang, Xingwei Qu, Ge Zhang, Yujia Qin, Baoquan Zhong, Chengquan Jiang, Jinxin Chi, and Wanjun Zhong. 2025. ReTool: Reinforcement Learning for Strategic Tool Use in LLMs. arXiv:2504.11536 [cs.CL] `https://arxiv.org/abs/2504.11536`

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. 2020. Constructing a Multi-hop QA Dataset for Comprehensive Evaluation of Reasoning Steps. In *Proceedings of the 28th International Conference on Computational Linguistics (COLING)*. 6609–6625.

Jiajie Jin, Yutao Zhu, Xinyu Yang, Chenghao Zhang, and Zhicheng Dou. 2024. FlashRAG: A Modular Toolkit for Efficient Retrieval-Augmented Generation Research. *arXiv preprint arXiv:2405.13576* (2024).

Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data* 7, 3 (2019), 535–547.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP)*.

Xiaoxi Li, Guanting Dong, Jiajie Jin, Yuyao Zhang, Yujia Zhou, Yutao Zhu, Peitian Zhang, and Zhicheng Dou. 2025. Search-o1: Agentic Search-Enhanced Large Reasoning Models. *arXiv preprint arXiv:2501.05366* (2025).

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2024. Let's Verify Step by Step. In *The Twelfth International Conference on Learning Representations (ICLR)*.

Heng Lin and Zhongwen Xu. 2025. Understanding Tool-Integrated Reasoning. *arXiv preprint arXiv:2508.19201* (2025).

Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A. Smith, and Mike Lewis. 2023. Measuring and Narrowing the Compositionality Gap in Language Models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*. 5687–5711.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. 2023. Direct Preference Optimization: Your Language Model is Secretly a Reward Model. *arXiv preprint arXiv:2305.18290* (2023).

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language Models Can Teach Themselves to Use Tools. In *Advances in Neural Information Processing Systems (NeurIPS)*.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. *arXiv preprint arXiv:1707.06347* (2017).

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models. *arXiv preprint arXiv:2402.03300* (2024).

Qwen Team. 2024. Qwen2.5 Technical Report. *arXiv preprint arXiv:2412.15115* (2024).

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2022. MuSiQue: Multihop Questions via Single-hop Question Composition. *Transactions of the Association for Computational Linguistics* 10 (2022), 539–554.

Liang Wang, Nan Yang, Xiaolong Huang, Binxing Jiao, Linjun Yang, Daxin Jiang, Rangan Majumder, and Furu Wei. 2022. Text Embeddings by Weakly-Supervised Contrastive Pre-training. *arXiv preprint arXiv:2212.03533* (2022).

Jialong Wu, Wenbiao Yin, Yong Jiang, Zhenglin Wang, Zekun Xi, Runnan Fang, Deyu Zhou, Pengjun Xie, and Fei Huang. 2025. WebWalker: Benchmarking LLMs in Web Traversal. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (ACL)*.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A Dataset for Diverse, Explainable Multi-hop Question Answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2369–2380.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In *The Eleventh International Conference on Learning Representations (ICLR)*.

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyan Luo, Zhangchi Feng, and Yongqiang Ma. 2024. LlamaFactory: Unified Efficient Fine-Tuning of 100+ Language Models. *arXiv preprint arXiv:2403.13372* (2024).

A Implementation Details

A.1 Repository layout

The companion repository `tool_star_aspo/` is organized into three work streams: `dataset_analysis/` (scripts and reports for parsing the SFT and RL corpora and the model-generated traces), `evaluation/` (`scoring.py` for cheap metrics and `modal_judge.py` for the 72B LLM-as-judge), and `aspo/` (the ASPO algorithm, to be populated).

A.2 Tavily adapter

The Tavily adapter rewrites only the body of `bing_web_search` in Tool-Star's `bing_search.py`, preserving the function signature and the `{webPages:{value:...}}` return shape. All downstream consumers (`extract_relevant_info`, the asynchronous variant, the `deep_search` wrapper) are therefore unchanged. The patch is applied at container start, so the upstream repository on disk is never modified. The same patch is also applied inside the RL trainer's vendored copy at `Tool_Star_RL/src/verl/verl/workers/rollout/vllm_rollout/web_search/bing_search.py`, keeping train- and eval-time web search behavior identical.

A.3 FlashRAG service architecture

Tool-Star's paper-faithful Wikipedia retrieval relies on the FlashRAG toolkit Jin et al. (2024): an e5-base-v2 Wang et al. (2022) encoder embeds the query, and a FAISS Johnson et al. (2019) flat inner-product index retrieves the top-k passages from the `wiki18_100w` corpus (Wikipedia 2018 chunked into 100-word passages, 21,015,324 rows in total). We host this as a Modal cloud service in three pieces:

- A Modal Volume (`tool-star-flashrag`) carries the three large artifacts that must co-exist with the service: the encoder snapshot ($\sim$440 MB), the FAISS index built with `faiss_index_type=Flat` ($\sim$32 GB), and the corpus JSONL ($\sim$6 GB).
- A single `@modal.web_server(port=1243)`-decorated function (`flashrag_service`) runs on `H200:1` (141 GB VRAM). On startup it spawns `host_wiki.py` from `Tool-Star/evaluation/search/`, which loads the FAISS index onto GPU 0, loads the encoder, and exposes `/health`, `/search` (single-query), and `/batch_search` (list-of-queries) endpoints. Cold-start boot is $\sim$7–10 min (corpus load $\sim$1.5 min, FAISS GPU transfer $\sim$5–7 min).
- Modal returns a public HTTPS URL of the form `https://tool-star-rl-flashrag-service-dev.modal.run`. Both the evaluation pipeline (`modal_flashrag.py::eval_qa`) and the RL training pipeline (`modal_rl.py::train_rl`, via the `search_url` Hydra override) reach the service over this URL.

The H200 choice (over H100) is empirically motivated: `faiss-gpu`'s copy of the $\sim$32 GB index alongside vLLM during evaluation exceeded H100's 80 GB; B200 (192 GB) was the natural alternative but vLLM 0.6.3 / torch 2.4 lack Blackwell (`sm_100`) kernels and crash with "no kernel image available" on B200, so we run vLLM on H200 at `gpu_memory_utilization=0.60` to leave room for the FAISS index ($141 \times 0.60 = 85$ GB for vLLM, 32 GB for FAISS, 24 GB safety margin).

A.4 Patched `utils.py`

Tool-Star's `evaluation/utils.py` defines `search()` and `batch_search()` stubs that originally pointed at the paper authors' internal Wikipedia service. We replace this file with a vendored `utils_patched.py` at container start (via `shutil.copy`, never editing the upstream repo on disk). Our version routes `search()` to `POST /search` (single-query payload `{query, top_n}`), `batch_search()` to `POST /batch_search` (`{query: List[str], top_n}`), and enforces a 60-second per-call timeout (the upstream default of `None` blocks forever if the service is unreachable). Each call emits a single-line stderr trace (`[search] CALL q=...`, `[search] OK status=200 latency=85ms`), which appears in the Modal task log and is invaluable for confirming that retrieval is being exercised during long sweeps.

A.5 Service validation

Before integrating the service into the eval and training pipelines we ran a standalone smoke test (`modal_flashrag.py::test_flashrag_solo`) that boots the service, issues three representative queries ("Who was the first president of the United States?", "What year did the Eiffel Tower complete construction?", and a physics question), and prints the top-3 retrieved passage titles and per-query latency. The third query takes $\sim$1.9 s cold (encoder pre-warmup) and subsequent queries are served in $\sim$85 ms (GPU FAISS, top-5). The test exercises every component the eval and training pipelines depend on and is documented in `evaluation/flashrag/modal_flashrag.py`.

A.6 Reproducing the headline numbers

1. Generate the 100 traces:

```
modal run evaluation/modal_inference.py::main --dataset gsm8k --counts 30
modal run evaluation/modal_inference.py::main --dataset math500 --counts 20
modal run evaluation/modal_inference.py::main --dataset webwalker --task qa \
    --prompt-type code_search --counts 50
```

2. Run the LLM-as-judge over math traces:

```
modal run evaluation/modal_judge.py::main \
    --traces-files traces/gsm8k_30ex.json,traces/math500_20ex.json
```

3. Generate the analysis report:

```
python dataset_analysis/scripts/analyze_traces.py \
    --files traces/gsm8k_30ex_judged.json \
            traces/math500_20ex_judged.json \
            traces/webwalker_50ex.json
```

A.7 The four modules of the ASPO package

Table 7: The four modules of the `aspo` package.

Module	Lines	Role
`aspo/advantage.py`	151	`compute_aspo_advantage(...)`: the algorithmic core. Computes A_{GRPO} via group normalization, then adds $\lambda\phi$ with the three terms of Equation 2. Returns a (B, L) advantage tensor identical in shape to GRPO's.
`aspo/detection.py`	149	`detect_structural_features(...)`: per-trajectory bigram/offset detection over the response token-ID tensor. Uses single-token IDs for `<python>`, `</python>`, `<answer>`, `<think>`, `<search>` in the Qwen2.5 vocabulary.
`aspo/integration.py`	124	`patch_ray_trainer(...)`: idempotent in-place edit that inserts an `elif adv_estimator == 'aspo':` branch into VERL's `ray_trainer.py::compute_advantage()`. The edit happens at container start, never on disk in the source-of-truth checkout.
`aspo/__init__.py`	20	Re-exports the four public symbols above.

B SFT Training Details

We document four staged experiments performed to validate the cold-start SFT pipeline before and during the paper-faithful run. The following section gives full details of this training and the

Table 8: Resource consumption summary across the SFT pilots and the inference passes used to characterize the resulting checkpoints. The *Training* block reports the wall-clock of each `modal_sft.py` run; the *Inference* block aggregates the per-checkpoint trace generation and LLM-as-judge passes that produced the headline tables of this report.

Experiment	Samples	Epochs	GPUs	Wall-clock
Training				
SFT smoke (Pilot 1)	200	1	8×H100	∼10 min
SFT medium (Pilot 2)	5,000	1	8×H100	3m38s
SFT full (Pilot 4)	53,971	3	8×H100	1h35m12s
Inference and evaluation				
Trace generation, 30 GSM8K (per checkpoint)	—	—	1×A10G	∼3–5 min
LLM-as-judge, 50 math traces (Qwen2.5-72B)	—	—	4×H100	∼10 min
WebWalker, 50 traces (Tavily-backed)	—	—	1×A10G	∼25 min
Total for the entire campaign		∼3 h compute		

reasoning behind selecting epoch 2 for future comparisons in the paper. All training pilots use the same backbone (Qwen2.5-3B-Instruct), the same DeepSpeed ZeRO-3 + FlashAttention-2 stack, and the same Modal serverless deployment (`tool_star_aspo/training/sft/modal_sft.py`). They differ in sample count, epoch count, and—for the inference-time pilots—in what checkpoint is being evaluated.

B.1 Pilot 1: Pipeline Smoke Test (200 samples, 1 epoch)

The smoke pilot is designed to validate the end-to-end pipeline (image build, dataset registration, DeepSpeed bootstrap, ZeRO-3 sharding across 8 H100s, WandB logging, checkpoint persistence to a Modal Volume) with minimal compute. The model is not expected to be useful for inference.

Table 9: SFT smoke configuration. The same 8×H100 layout as the full run is used to also validate distributed correctness.

Parameter	Value
Samples	200 (first slice of SFT-54K)
Epochs	1
GPUs	8×H100 80GB
Effective batch size	8 (`per_device=1` × `accum=1` × 8)
Steps	25
Wall-clock	∼10 min (first run, including image build)

Observations: `train/loss` drops from ≈1.63 to ≈1.20 over the 25 steps. The cosine learning rate schedule, the warmup, and the gradient clipping all behave as configured (verified in WandB). Critically, `torch.cuda.device_count() = 8` confirms Modal allocated the requested 8 GPUs. The smoke validates the *pipeline*; it does not validate the *model*.

B.2 Pilot 2: SFT Medium, throughput and Loss Calibration (5,000 samples, 1 epoch)

The medium pilot produces a usable but undertrained checkpoint, allowing us to measure (i) per-step throughput on H100, (ii) the shape of the loss curve at the effective batch size of 128 used by the paper, and (iii) the first `eval/loss` on held-out data.

Two findings: First, `eval/loss` (0.962) is lower than the averaged `train/loss` (1.025). This is consistent and expected—the training loss is averaged across all 38 steps, including the high-loss early ones, while the evaluation loss reflects the converged model. The gap is small enough that we observe no overfitting on the 5K subsample. Second, the measured per-step time of ≈5.74 s on 8×H100 allows us to revise the full-run estimate downward: 1,266 steps for the full configuration

projects to ≈2 h of wall-clock, significantly shorter than the original ∼1 day on 8×A100 reported in the paper.

B.3 Pilot 3: Inference Validation against the Published Checkpoint

To validate that the medium checkpoint produces well-formed multi-tool trajectories before the full training run, we deploy the checkpoint via vLLM and generate 30 GSM8K rollouts using the same pipeline that produced the 100-trace pilot. Outputs are compared against the published `Tool-Star-Qwen-3B` checkpoint on the same 30 prompts.

Table 10: SFT medium configuration and outcomes. Effective batch size matches the paper (128) via `per_device=4` × `accum=4` × 8 GPUs.

Parameter / Outcome	**Value**
Samples	5,000
Epochs	1
Effective batch size	128
Steps	38
Wall-clock training	218 s (≈3.6 min)
Throughput	22.5 samples/s, 0.17 steps/s
Final `train/loss` (averaged)	1.025
`eval/loss` (100 held-out)	0.962

Table 11: Pilot 3 results: SFT-medium vs. published `Tool-Star-Qwen-3B` on GSM8K ($n = 30$). Despite using ∼9% of the SFT data and 1/3 of the epochs and no RL phase, the medium checkpoint recovers ∼91% of the baseline's accuracy.

Model	**Training**	n	P**(correct)**
Tool-Star-Qwen-3B (public)	54K × 3 + RL	30	0.733
SFT-medium (ours)	5K × 1	30	0.667
Difference			−6.7 pts

Crucially, format compliance is strong: 27 of 30 SFT-medium trajectories (90%) emit a well-formed block and close with the required `\boxed{...}` structure, with the remaining 3 cases (10%) being "no-tools" chain-of-thought outputs that match the 9.3% no-tools rate present in the SFT corpus. Zero of the medium-checkpoint trajectories combine Python and search, again consistent with the structural absence of mixed-tool examples in the SFT data. The accuracy and category distributions are summarized graphically in Figures 5 and 6.

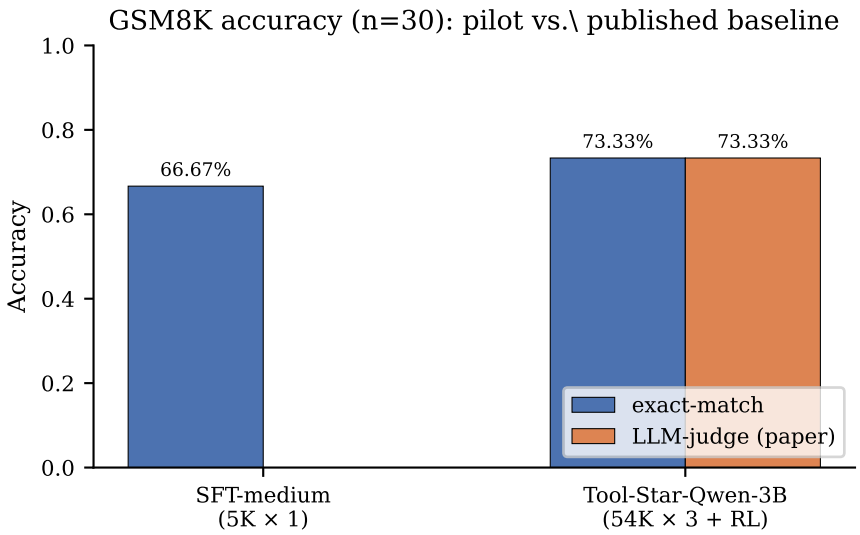

Figure 5: GSM8K accuracy on the 30-prompt inference pilot

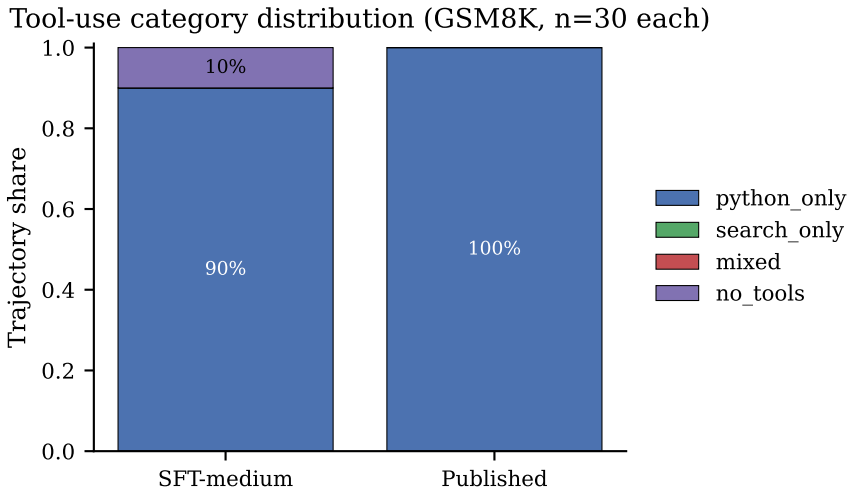

Figure 6: Tool-use category distribution (GSM8K, n=30 each)

For our ASPO design, the bigram contrast we identified in Section 3 reproduces in the medium checkpoint's outputs: across the combined 60 rollouts (30 medium + 30 published), $\langle$python $\rightarrow$ answer$\rangle$ achieves $P(\text{correct}) = 0.796$ versus $\langle$python $\rightarrow$ think$\rangle$ at 0.091, an effect size of $\sim 8\times$ that is robust to which of the two models generated the trajectory.

B.4 Decision: Proceed to the Full Run

The pilots collectively support proceeding to the full paper-faithful recipe (`configs/full.yaml`: 54K $\times$ 3 epochs) without further configuration changes. The revised wall-clock estimate of ~ 2 h is well within budget for the project, and the -6.7-point gap on GSM8K relative to the published baseline plus RL-phase improvements is consistent with the expected contribution of the additional data and epochs.

B.5 Full Run and Epoch-Wise Checkpoint Selection

We executed the full paper-faithful recipe and evaluated the three saved epoch checkpoints individually on the same 30 GSM8K prompts used for Pilots 1–3. Two observations from Table 12. First, epoch 2 (`checkpoint-828`) achieves the highest accuracy in this sample, with epoch 3 regressing by 3.3 points—suggestive of mild overfitting at the third pass through SFT-54K. We therefore retain `checkpoint-828` as the canonical SFT initialization for both the Tool-Star RL baseline and the ASPO RL variant in subsequent experiments. Second, the apparent superiority of our SFT-only checkpoint over the publicly-released SFT+RL checkpoint on GSM8K is intriguing but not conclusive at $n = 30$.

Table 12: Per-epoch inference accuracy on the same 30 GSM8K prompts. All checkpoints emit well-formed multi-tool trajectories (98% Python, 2% no-tools). Metric is exact-match after normalization; for the public baseline, exact-match coincides with the paper's LLM-judge for this sample. Despite using only SFT (no RL), `checkpoint-828` narrowly exceeds the publicly released Tool-Star-Qwen-3B on this subset, though the comparison falls inside the $\pm$17-point binomial 95% CI at $n = 30$.

Model	**Data**	n	P**(exact)**
SFT-medium (`sft-medium`)	5K $\times$ 1	30	0.667
SFT-full epoch 1 (`checkpoint-414`)	53K $\times$ 1	30	0.700
SFT-full epoch 2 (`checkpoint-828`)	53K $\times$ 2	30	**0.833**
SFT-full epoch 3 (`checkpoint-1239`)	53K $\times$ 3	30	0.800
Tool-Star-Qwen-3B (published)	54K $\times$ 3 + RL	30	0.733

Table 13: Outcomes of the full run. Wall-clock came in under the projection; `eval/loss` dropped sharply across epochs, with no overfitting (`eval/loss` $<$ `train/loss` at every measurement).

Parameter / Outcome	**Value**
Samples	53,971 (full SFT-54K, -2% held-out)
Epochs	3
Steps	1,239
Wall-clock training	1 h 35 min (95 min)
Throughput	27.8 samples/s, 0.22 steps/s
Final `train/loss` (averaged)	0.732
Final `eval/loss` (1,080 held-out)	0.650
Checkpoints retained	`checkpoint-414`, `-828`, `-1239`

B.6 SFT Training Curves

Figure 7 shows the loss trajectories of the full paper-faithful run. Panel (a) plots the per-step training loss across all 1,239 steps, with the three epoch boundaries (checkpoints) marked. The loss falls steeply during warmup, settles into the characteristic noisy plateau of token-level cross-entropy, and continues a gradual descent across epochs. Panel (b) plots the held-out evaluation loss measured at each checkpoint: it drops from 0.6576 at epoch 1 to a minimum of 0.6471 at epoch 2, then ticks back up to 0.6497 at epoch 3.

This non-monotone evaluation curve is the clearest signal for checkpoint selection. While the evaluation loss never exceeds the training loss—so there is no severe overfitting—the upturn at epoch 3 marks the onset of mild overfitting on the third pass through SFT-54K, consistent with the 3.3-point accuracy regression reported in Section 12. We therefore select epoch 2 (`checkpoint-828`) as the canonical SFT initialization for all subsequent RL experiments.

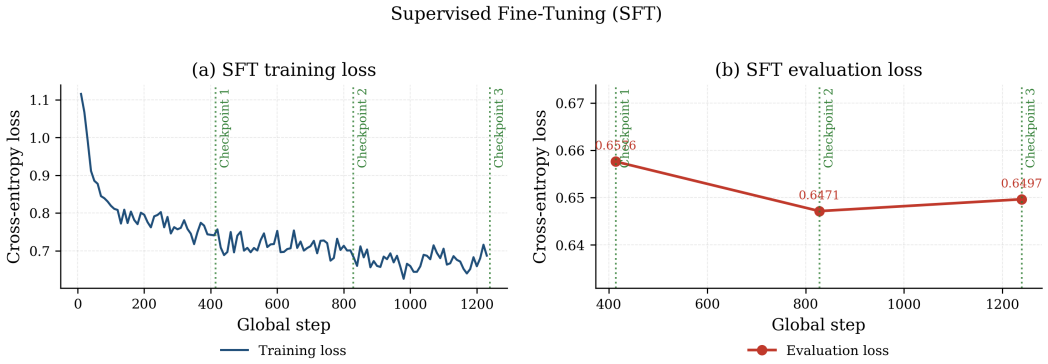

Figure 7: Supervised fine-tuning loss curves. (a) Per-step training loss over the full 3-epoch run; vertical lines mark epoch-boundary checkpoints. (b) Held-out evaluation loss at each checkpoint, reaching its minimum at epoch 2 (`checkpoint-828`) before rising at epoch 3—the signature that motivates selecting epoch 2.

C Training Figures: ASPO vs. Tool-Star (GRPO) at 30 Steps

(a) Tool-Star `critic/score/mean`

(b) ASPO `critic/score/mean`

(c) Tool-Star `actor/loss`

(d) ASPO `actor/loss`

(e) Tool-Star `response_length/mean`

(f) ASPO `response_length/mean`

Figure 8: Training-time telemetry for the matched-compute runs. Left: Tool-Star GRPO baseline (steps 1–30). Right: our ASPO run (steps 16–30). Rows: `critic/score/mean`, `actor/loss`, and `response_length/mean`.